```
Esercizio 1 - Soluzione
(a)
 Non-Modificabile perche le operazioni pubbliche sono solo osservatori
 (predicati: isEmpty, isRoot, hasSons e operazioni getLeaf e sons)

(b)
 public class AlberoADT<A> implements AlberoAPI<A>, Cloneable{
 private A root;
 private Vector<AlberoADT<A>> sons;
 private int size;

(c,e)
 public AlberoADT(){}
 public AlberoADT(A r, LinkedList<AlberoADT<A>> sonList){
    if (r == null) return;
    root = r; size = 1;
    if (sonList == null) return;
    sons = new Vector<AlberoADT<A>>();
    for (int i = 0; i<sonList.size();i++){
        AlberoADT<A> temp = sonList.get(i);
        sons.add(i,temp);
        size = size + temp.size;
    }
 }
 public getSize(){
    return size;
 }

(d)
 AF(c) = [] if c.root == null
 AF(c) = [V] if c.root != null && V == c.root && c.sons == null
 AF(c) = [V - T1,...,Tn] if c.root != null && V == c.root
                            && c.sons != null
                            && n == c.sons.size()
                            && (1<=i<=n), AF(c.sons.get(i-1)) = Ti
 I(c) = c.root == null IFF (c.sons == null & c.size == 0)
        && if c.root != null
           then Either (c.sons == null & c.size == 1)
                Or (0<=i<c.sons.size())
                            c.size = 1 + ∑(c.sons.get(i).getSize())


Esercizio 2 - Soluzione
(a)
module type TREEAPI =
sig
    type 'a tree
    val mkE: () -> 'a tree
    val mkR: int -> 'a -> ('a tree) list
    val isEmpty: 'a tree -> bool
    val isRoot: 'a tree -> bool
    val hasSons: 'a tree -> bool
    val getLeaf:  'a tree -> 'a
    val sons: 'a tree -> ('a tree) list
end;;

(b)
exception IllegalArgsIn of string;;
module TreeADTE =
(struct
```

```
    type 'a tree = 'a TreeADT.tree
    let mkE () = TreeADT.mkE()
    let mkR k r tt = (* vedi nota 1 *)
            (let zz = List.map (fun t -> getSize t) tt
             in let rec sum ll = (match ll with [] -> 0
                                  |n::ns -> n+(sum ns))
                in let n = 1 + sum zz
                   in if n > k then raise (IllegalArgsIn "mkR")
                      else TreeADT.mkR r tt
             )
    let isRoot t = TreeADT.isRoot t
    let getSize t = TreeADT.getSize t
    ....
end)

Esercizio 3 - Soluzione
public class MuQSingleADTXEE<A> extends MuQSingleADTXE<A>{
    private int maxSize;
    //metodi
    public MuQSingleADTXEE(int k){// vedi nota 2
        maxSize = k;
    }
    public void add(A x){// vedi nota 3
        if (size()< maxSize) super.add(x);
    }
    //additionals
    //equals ereditata da Object
    //toString ed elements ereditate dalla super
    //clone ereditata dalla super oppure:
    public MuQSingleADTXEE<A> clone(){
        return (MuQSingleADTXEE<A>) super.clone();
    }
}

NOTA
1) TreeADTE e` una specializzazione di TreeADT (infatti riusa valori e codice di TreeAD
   ma non e` richiesto che sia un ADT per TREEAPI (infatti la segnatura di mKR e` diffe
   da quella indicata in TREEAPI, allo stesso tempo il corpo (struct...end) del modulo
   TreeADTE non specifica alcuna API implementata)
2) In java, i costruttori non hanno segnatura API e mKR diventa il nuovo costruttore pe
   code non vuote.
3) Se la coda contiene meno di k elementi allora super.add modifica correttamente. In c
   contrario (i.e. esattamente k) la coda e` invariata.
```